Wild Card: Traveling Sales Person

Abstract: The final project of the semester. The most challenging part of this problem for me was getting the proper fitness metric. Originally, I had worked on a rhythm-based music GA but had issues with that fitness metric as well. Overall, this assignment was fun but the wide variety of possibilities in it made it a challenge.

Task 1: Rng-Tour

Task 1 Demo:

[3]> *cities*

(A B C D E F G H I J)

[4]> (get 'a 'b)

600

[5]> (get 'c 'h)

750

[6]> (symbol-plist 'i)

(A 825 B 650 C 600 D 900 E 400 F 750 G 1000 H 1150 J 1050)

[7]> (rng-tour)

(F D H E C G B I A J F)

[8]> (rng-tour)

(H E C G J B F I A D H)

[9]> (rng-tour)

(I E D B C A H J G F I)

Task 1 Code:

```
;-------------------------------------------------------
; Task 1 - Tour setup


( setf *cities* '( a b c d e f g h i j ) )

;City distances
```

```
( setf ( symbol-plist 'a ) '(b 600 c 500 d 450 e 650 f 700 g 900 h 750 i 825 j
1000) )

( setf ( symbol-plist 'b ) '(a 600 c 700 d 950 e 1100 f 800 g 1150 h 800 i 650 j
975) )

( setf ( symbol-plist 'c ) '(a 500 b 700 d 1100 e 475 f 900 g 1200 h 750 i 600 j
800) )

( setf ( symbol-plist 'd ) '(a 450 b 950 c 1100 e 1200 f 550 g 650 h 650 i 900 j
1000 ) )

( setf ( symbol-plist 'e ) '(a 650 b 1100 c 475 d 1200 f 800 g 1000 h 950 i 400 j
600 ) )

( setf ( symbol-plist 'f ) '(a 700 b 800 c 900 d 550 e 800 g 1200 h 1350 i 750 j
650 ) )

( setf ( symbol-plist 'g ) '(a 900 b 1150 c 1200 d 650 e 1000 f 1200 h 500 i 1000
j 800 ) )

( setf ( symbol-plist 'h ) '(a 750 b 800 c 750 d 650 e 950 f 1350 g 500 i 1150 j
850 ) )

( setf ( symbol-plist 'i ) '(a 825 b 650 c 600 d 900 e 400 f 750 g 1000 h 1150 j
1050 ) )

( setf ( symbol-plist 'j ) '(a 1000 b 975 c 800 d 1000 e 600 f 650 g 800 h 850 i
1050 ) )

( defun rng-tour ()
    ( setf plan ( rng-tour-help *cities* ) )
    ( snoc ( car plan ) plan )
)

( defmethod rng-tour-help (li)
    ( cond
        ( ( null li ) () )
        (t
            ( setf cities ( nth ( random ( length li ) ) li ) )
            ( setf li ( remove cities li :count 1 ) )
            ( cons cities ( rng-tour-help li ) )
        )
    )
)
```

Task 2: Tour Mutation

Task 2 Demo:

[16]> (setf h (rng-tour))

(C D H J A B E G I F C)

[17]> (mutation h)

(C D I J A B E G H F C)

[18]> (mutation h)

(C D H J A B E G F I C)

[19]> (mutation h)

(C D H F A B E G I J C)

[20]> (mutation h)

(C F H J A B E G I D C)

[21]> (mutation h)

(C D H J A E B G I F C)

[22]> (mutation h)

(C D H J A B E F I G C)

Task 2 Code:

```
;-------------------------------------------------------
; Task 2 mutation - plan changing

( defun mutation ( tour &aux city1 city2 point1 point2 plan)
    ( setf point1 ( position ( pick ( cdr ( rdc tour ) ) ) tour ) )
    ( setf point2 ( position ( pick ( cdr ( rdc tour ) ) ) tour ) )
    ( cond
        ( ( = point1 point2 )
            ( mutation tour )
        )
        ( t
            ( setf plan ( copy-tree tour ) )
            ( setf city1 ( nth point1 plan ) )
            ( setf city2 ( nth point2 plan ) )
            ( setf ( nth point1 plan ) city2 )
            ( setf ( nth point2 plan ) city1 )
```

```
        plan
        )
    )
)
```

Task 3: Crossover

Task 3 Demo:

[7]> (crossover (rng-tour) (rng-tour))

(B D F A C E G I J H B)

[8]> (crossover (rng-tour) (rng-tour))

(E G J H F C B I A D E)

[9]> (crossover (rng-tour) (rng-tour))

(A C B J I D F G H E A)

[10]> (crossover (rng-tour) (rng-tour))

(C G F E H A B D I J C)

[11]> (crossover (rng-tour) (rng-tour))

(A I E B G H F J C D A)

[12]> (crossover (rng-tour) (rng-tour))

(H C E G J A F I D B H)**Task 3 Code:**

```
;-----------------------------------------------------------
; Task 3 - Crossover

( defmethod crossover ( ( m list ) ( f list ))
    ( setf sim-cities ( get-sim-cities m f ) )
    ( cond
        ( ( null sim-cities ) nil )
        (t
            ( setf inter ( intersection m sim-cities ) )
            ( setf difference ( - ( length m ) ( length ( member ( car inter ) m
) ) ) )
            ( setf new-plan ( append ( first-n m difference ) sim-cities ( rest-n
m ( + difference ( length sim-cities ) ) ) ) )
            new-plan
        )
    )
)
```

```
( defmethod get-sim-cities ( ( m list ) ( f list ) )
    (setf 5city (common-cities (rdc (cdr m)) (rdc (cdr f)) 5 ))
    (setf 4city (common-cities (rdc (cdr m)) (rdc (cdr f)) 4 ))
    (setf 3city (common-cities (rdc (cdr m)) (rdc (cdr f)) 3 ))
    ( cond
        ( ( not ( null 5city ) ) 5city )
        ( ( not ( null 4city ) ) 4city )
        ( ( not ( null 3city ) ) 3city )
        (t nil)
    )
)

(defun common-cities (m f n)
    (cond
        ((< (length m) n) nil)
        ((common-cities-help m f n) (common-cities-help m f n) )
        (t
            (common-cities (cdr m) f n)
        )
    )
)

(defun common-cities-help (m f n)
    (cond
        ((< (length f) n) nil)
        ((= n (length (intersection (first-n m n) (first-n f n))))
            (intersection (first-n m n) (first-n f n)))
        (t
            (common-cities-help m (cdr f) n)
        )
    )
)


(defun first-n (l p)
    (first-n-help l p '() 0)
)

(defun first-n-help (l p res n)
    (cond ((eq n p) res)
        (t
            (first-n-help (cdr l) p (append res (list (car l))) (+ n 1))
        )
    )
)
```

```
(defun rest-n (l p)
    (rest-n-help l p 0)
)

(defun rest-n-help (l p n)
    (cond ((eq n p) l)
        (t
            (rest-n-help (cdr l) p (+ n 1))
        )
    )
)
```

Task 4: Mutation/Crossover Demo

Task 4 Demo:

[23]> (mutation-demo)

s = (A D G E B C I J H F A)

m = (A D G C B E I J H F A)

s = (A D G E B C I J H F A)

m = (A D G B E C I J H F A)

s = (A D G E B C I J H F A)

m = (A D G H B C I J E F A)

s = (A D G E B C I J H F A)

m = (A F G E B C I J H D A)

s = (A D G E B C I J H F A)

m = (A D G F B C I J H E A)

s = (A D G E B C I J H F A)

m = (A D F E B C I J H G A)

s = (A D G E B C I J H F A)
m = (A D G E B C F J H I A)

s = (A D G E B C I J H F A)
m = (A D G B E C I J H F A)

s = (A D G E B C I J H F A)
m = (A D G E C B I J H F A)

s = (A D G E B C I J H F A)
m = (A D G E B F I J H C A)

NIL
[24]> (crossover-demo)
m = (B G F E H A J D I C B)
x = (B F J E D I C H G A B)
f = (B F J E C D I H G A B)

m = (A E G J D F I B C H A)
x = (H A G E F J I B C D H)
f = (H A G E F J C I B D H)

m = (C H D G I A B J F E C)
x = (C H D G I A B J F E C)
f = (G F A E C D H I B J G)

m = (J G B C F D I H E A J)
x = (I J H G B C F D E A I)

f = (I J H G D F B C E A I)

m = (H I B G A F E J C D H)

x = (H I B G A F E J C D H)

f = (H A D E B C G J F I H)

m = (J D G B A F C E H I J)

x = (D H A I B J G F C E D)

f = (D H A I B J G C E F D)

m = (D C G J E A H B I F D)

x = (D C G J E A H B I F D)

f = (E F A H J I C G D B E)

m = (G H B F A J E C I D G)

x = (G J E C H A I D B F G)

f = (G C E J H A I D B F G)

m = (I A G B E F D J H C I)

x = (I A G B E F D J H C I)

f = (D A I G J E H C F B D)

m = (G C D F B J A I H E G)

x = (G C D F B J A I H E G)

f = (A J E F G H I D B C A)

Task 4 Code:

```
;-------------------------------------------------------------
; Task 4 -Mutation/Crossover Demo

( defun mutation-demo ()
```

```
    ( setf c ( rng-tour *cities* ) )
    ( dotimes ( i 10 )
        ( format t "s = ~A~%" c )
        ( setf m ( mutation c ) )
        ( format t "m = ~A~%~%" m )
    )
)

(defun crossover-demo ()
    (setf m ( rng-tour *cities* ) )
    (setf f ( rng-tour *cities* ) )
    ( dotimes ( i 10 )
        ( format t "m = ~A~%" m )
        ( setf x ( crossover m f ) )
        ( format t "x = ~A~%" x )
        ( format t "f = ~A~%~%" f)
        (setf m ( rng-tour *cities* ) )
        (setf f ( rng-tour *cities* ) )
    )
)
```

Task 5: Fitness Metric

Task 5 Demo:

[25]> (fitness-demo)

plan = (J F B H A D E I G C J)

fitness = 2750

plan = (C E F D H A I G B J C)

fitness = 2275

plan = (B A I F D C E H J G B)

fitness = 1850

plan = (I H E B A C D G F J I)

fitness = 2650

plan = (C D F J H G I E B A C)

fitness = 2400

plan = (F C H A I J B G E D F)

fitness = 2550

plan = (E D H J G I F A C B E)

fitness = 2200

plan = (I H C F D B E J A G I)

fitness = 2400

plan = (B I D E J F C G A H B)

fitness = 2400

plan = (I A G J C E H F B D I)

fitness = 2650

Task 5 Code:

```
;-----------------------------------------------------------
; Task 5 - Fitness Metric

( defmethod fitness ( li )
    ( setf li ( rdc ( cdr li ) ) )
    ( cond
        ( ( null ( cdr li ) ) 0 )
        ( t
            ( + ( get ( car li ) ( cadr li ) )
                ( fitness ( cdr li ) ) )
        )
    )
)

( defun fitness-demo (&aux plan distance)
    ( setf plan ( rng-tour *cities* ) )
```

```
    ( dotimes ( i 10 )
        ( setf distance ( fitness plan ) )
        ( format t "plan = ~A~%" plan )
        ( format t "fitness = ~A~%~%" distance )
        ( setf plan (rng-tour *cities* ) ) )
    )
)
```

Task 6: Individual Class

Task 6 Demo:

[26]> (individual-demo)

0 (F I E A H B D G C J F) 2100

1 (I B A G C E D H J F I) 3000

2 (I H G D F J C E B A I) 1850

3 (H J G F C E D B I A H) 2900

Fitness of i1 = 3000

Fitness of i2 = 1850

Fitness of i3 = 2900

Task 6 Code:

```
;-----------------------------------------------------------
; Task 6 - Individual Class

( defclass individual ()
    (
        ( route :accessor individual-route :initarg :route )
        ( fitness :accessor individual-fitness :initarg :fitness )
        ( number :accessor individual-number :initarg :number )
    )
)

( defmethod random-individual ( &aux plan )
    ( setf plan ( rng-tour ) )
    ( make-instance 'individual
        :route plan
        :fitness ( fitness plan )
        :number 0
    )
)
```

```
( defmethod new-individual ( ( nr number ) ( notes list ) )
    ( make-instance 'individual
        :route notes
        :fitness ( fitness notes )
        :number nr
    )
)

( defmethod display ( ( i individual ) )
    ( display-nnl i ) ( terpri )
)

( defmethod display-nnl ( ( i individual ) )
    ( prin1 ( individual-number i ) )
    ( princ ( filler ( individual-number i ) ) )
    ( prin1 ( individual-route i ) )
    ( princ " " )
    ( prin1 ( individual-fitness i ) )
    ( princ ( filler ( individual-fitness i ) ) )
)

( defmethod filler ( ( n number ) )
    ( cond
        ( ( < n 10 ) "     " )
        ( ( < n 100 ) "    " )
        ( ( < n 1000 ) "   " )
        ( ( < n 10000 ) "  " )
        ( ( < n 100000 ) " " )
    )
)


( defmethod individual-demo ( &aux i0 i1 i2 i3 one two three )
    ( setf i0 ( random-individual ) )
    ( display i0 )
    ( setf one ( rng-tour ) )
    ( setf i1 ( new-individual 1 one ) )
    ( display i1 )
    ( setf two ( rng-tour ) )
    ( setf i2 ( new-individual 2 two ) )
    ( display i2 )
    ( setf three ( rng-tour ) )
    ( setf i3 ( new-individual 3 three ) )
    ( display i3 )
```

```
    ( format t "Fitness of i1 = ~A~%" ( fitness one ) )
    ( format t "Fitness of i2 = ~A~%" ( fitness two ) )
    ( format t "Fitness of i3 = ~A~%" ( fitness three ) )
    nil
)
```

Task 7: Population Class

Task 7 Demo:

(population-demo)

Generation 0 population ...

1 (F G D E B C J H A I F) 2550

2 (G E C A I B D J H F G) 2250

3 (J F D G A B E I C H J) 2550

4 (A E I F G D C J H B A) 2700

5 (H I E J C G B A F D H) 2350

6 (J H D G F B I A E C J) 2500

7 (I B J H A F E D C G I) 2525

8 (D B C F I E H G A J D) 2400

9 (B C E F I G A H J D B) 2125

10 (F A G J H I D C E B F) 2650

11 (E H G I D F A B C J E) 2100

12 (D C F H A I G E J B D) 2650

13 (J A I F C B E D G H J) 2825

14 (D G B I C H E A F J D) 2700

15 (B F A H C E J D I G B) 2050

16 (J E H F A G D I C B J) 2300

17 (B H I J F G A E C D B) 2700

18 (D I H G A F E J C B D) 2850

19 (E I C J G A D B H F E) 1850

20 (F J H C D G I E B A F) 2950

21 (F A D J B G C E I H F) 2625

22 (J F E A I G C H B D J) 2825

23 (A B J G D F H C I E A) 2975

24 (I E H C B F J A D G I) 2300

25 (G C J D B F A E H I G) 2450

26 (D E A I J G B C H F D) 2850

27 (A C F H G I B D J E A) 2050

28 (H C B D J G E A F I H) 2700

29 (E C J I D B G A H F E) 2850

30 (H C F I E A J D B G H) 2300

31 (F D J E G B H A I C F) 2800

32 (C E A D J F B G H I C) 2450

33 (F G I D B H J E A C F) 2800

34 (E H D A F I C J G B E) 1950

35 (A E I J F D H B G C A) 1700

36 (H F B J I G E D C A H) 2850

37 (E G I D A F B H J C E) 2250

38 (I G A B H C D E J F I) 2800

39 (H I F J D A G E B C H) 2650

40 (I E G A C F D B J H I) 2050

41 (H B D C E G I A F J H) 2425

42 (F E I A H B C D G J F) 1850

43 (F H I A C B E D G J F) 2750

44 (D I A B H J E G F C D) 2225

45 (F E I G A H D C B J F) 1950

46 (A F B J I E C D G H A) 2325

47 (D A F I G J E H B C D) 2300

48 (A D G C E J I H B F A) 2175

49 (H D F E I B A J G C H) 1550

50 (H J B A D F G I C E H) 2625

51 (A D C G J E I B H F A) 2300

52 (B E H C D I F J A G B) 2800

53 (A F J D I H G C E B A) 2050

54 (C J G E B I F H A D C) 2650

55 (B F C H I D J G E A B) 3050

56 (H D I J B A E C F G H) 2525

57 (H A E B I F D G C J H) 1850

58 (E F H A I B G C D J E) 3325

59 (A E J C H I G F D B A) 2350

60 (J E B A H F C D I G J) 2750

61 (G A D H C J I E B F G) 2250

62 (J I G D H A E B F C J) 2300

63 (I E H C B G D F J A I) 2300

64 (J G C H A I B D F E J) 2600

65 (I E J C F D G A B H I) 2150

66 (B F D I H J C E G A B) 2500

67 (F C A E G B H D J I F) 2300

68 (G J E D C B A H F I G) 2300

69 (H F I J E A D C B G H) 1800

70 (B F E D G H C I A J B) 2200

71 (G E C A J I B H F D G) 2125

72 (B C D I J F H E G A B) 3500

73 (J E B F D H I G A C J) 2800

74 (D G I H F E C A B J D) 2825

75 (F A I C B J D G E H F) 2525

76 (A I F D J C B H G E A) 2450

77 (H C F I E A G B J D H) 2200

78 (D C H I E G F J B A D) 2350

79 (A I H J G D C B F E A) 3050

80 (G F C E I J H D B A G) 2150

81 (D F B J C A H E I G D) 2350

82 (F A G I J H C B D E F) 2700

83 (E C I J D G B F H A E) 2750

84 (I C E A H D F G J B I) 1775

85 (D A F H J B C I G E D) 2250

86 (J E F D C G H I A B J) 2400

87 (A I C F G J H E D B A) 2650

88 (C A H J D F I B G E C) 2500

89 (H D I E B A G F C J H) 2900

90 (A H C D I B J E G F A) 2625

91 (G D C H F A I E B J G) 3275

92 (C F B J A H G D I E C) 2300

93 (D B G A C H I J F E D) 2800

94 (E D J A C H B F G I E) 2300

95 (H B F I D E G A J C H) 2700

96 (G F D J E B I C A H G) 1800

97 (G I A D B F C H J E G) 2675

98 (B D A I F C E H G J B) 1675

99 (E A C I F D H G B J E) 1900

100 (C A H D F E B J I G C) 2400

average fitness = 2450.25

Sampling ...

*** - >=:

(#<INDIVIDUAL #x1AA7678D> #<INDIVIDUAL #x1AA76A15> #<INDIVIDUAL #x1AA765C5> #<INDIVIDUAL #x1AA76325>

#<INDIVIDUAL #x1AA763B5> #<INDIVIDUAL #x1AA76685> #<INDIVIDUAL #x1AA762DD> #<INDIVIDUAL #x1AA766E5>)

is not a real number

Task 7 Code:

```
;-------------------------------------------------
; Task 7 - The Population Class

( defconstant *population-size* 100 )
( defconstant *selection-size* 8 )


( defclass population ()
    (
        ( individuals :accessor population-individuals :initarg :individuals )
        ( generation :accessor population-generation :initform 0 )
    )
)

( defmethod size ( ( p population ) )
    ( length ( population-individuals p ) )
)

( defmethod display ( ( p population ) )
    ( terpri ) ( terpri )
    ( princ "Generation " )
    ( prin1 ( population-generation p ) )
    ( princ " population ..." )
    ( terpri ) ( terpri )
    ( dolist ( i ( population-individuals p ) )
        (display i )
```

```
    )
    ( terpri )
)

( defmethod initial-population ( &aux individuals )
    ( setf individuals () )
    ( dotimes ( i *population-size* )
        ( push ( new-individual ( + i 1 ) ( rng-tour ) ) individuals )
    )
    ( make-instance 'population :individuals ( reverse individuals ) )
)

( defmethod average ( ( p population ) &aux ( sum 0 ) )
    ( dolist ( i ( population-individuals p ) )
        ( setf sum ( + sum ( individual-fitness i ) ) )
    )
    ( / ( float sum ) *population-size* )
)

( setf *select-demo* nil )

( defmethod select-individual ( ( p population )
    &aux i candidates rn )
    ( setf candidates ( select-individuals p ) )
    ( setf mfi ( most-fit-individual candidates ) )
    ( if *select-demo* ( select-demo-helper candidates mfi ) )
    mfi
)

( defmethod select-individuals ( ( p population )
    &aux individuals candidates rn )
    ( setf individuals ( population-individuals p ) )
    ( setf candidates () )
    ( dotimes ( i *selection-size* )
        ( setf rn ( random *population-size* ) )
        ( push ( nth rn individuals ) candidates )
    )
    candidates
)

( defmethod most-fit-individual ( ( l list ) &aux max-value max-individual )
    ( setf max-individual ( car l ) )
    ( setf max-value ( individual-fitness max-individual) )
    ( dotimes ( i (- ( length l ) 1 ) )
        ( cond
```

```
            ( ( < ( individual-fitness (nth ( + 1 i ) l ) ) max-value )
                ( setf max-individual ( nth ( + 1 i ) l ) )
                ( setf max-individual ( individual-fitness max-individual) )
            )
        )
    )
    max-individual
)

( defmethod select-demo-helper ( ( l list ) ( i individual ) )
    ( princ "the sample of individuals ..." ) ( terpri )
    ( display l )
    ( terpri )
    ( princ "the most fit of the sample ... " ) ( terpri )
    ( display i )
    ( terpri )
    nil
)

( defmethod population-demo ( &aux p )
    ( setf p ( initial-population ) )
    ( display p )
    ( format t "average fitness = ~A~%~%" ( average p ) )
    ( setf *select-demo* t )
    ( format t "Sampling ...~%~%" )
    ( select-individual p ) ( terpri )
    ( format t "Sampling ...~%~%" )
    ( select-individual p ) ( terpri )
    ( format t "Sampling ...~%~%" )
    ( select-individual p ) ( terpri )
)
```

Task 8: Maybe Mutate

Task 8 Demo:

[29]> (mutate-demo)

0 (J G B H I D C F A E J) 3400

0 (J G B H I D F C A E J) 2850

0 (J G B D I H F C A E J) 3400

0 (J G B D I C F H A E J) 2950

0 (J G B H I C F D A E J) 3200

0 (J G B H F C I D A E J) 3100

0 (J G B H F E I D A C J) 2900

0 (J F B H G E I D A C J) 1700

0 (J F D H G E I B A C J) 1450

0 (J G D H F E I B A C J) 2400

0 (J G D H F E A B I C J) 2650

0 (J G D B F E A H I C J) 2100

0 (J G D C F E A H I B J) 2200

0 (J G D F C E A H I B J) 2200

0 (J G E F C D A H I B J) 2350

0 (J G E F C D A H B I J) 2350

0 (J G E F C D A H I B J) 2350

0 (J B E F C D A H I G J) 2450

0 (J G E F C D A H I B J) 2350

0 (J D E F C G A H I B J) 3000

0 (J D E F I G A H C B J) 2850

NIL

[30]> (maybe-mutate-demo)

0 (I H B E J D C F A G I) 2500

0 (I H B E J C D F A G I) 2500 *

0 (I H B E J C D F A G I) 2500

0 (I H B C J E D F A G I) 2800 *

0 (I H B C J E A F D G I) 2250 *

0 (I H B A J E C F D G I) 2275 *

0 (I H B A J E C F D G I) 2275

0 (I C B A J E H F D G I) 2650 *

0 (I E B A J C H F D G I) 2850 *

0 (I E B A J C G F D H I) 3300 *

0 (I E B J A C G F D H I) 3300 *

0 (I E B J A C G F D H I) 3300

0 (I E G J A C B F D H I) 2700 *

0 (I E G J A C B F D H I) 2700

0 (I J G E A C B F D H I) 2150 *

0 (I J G E A C B F D H I) 2150

0 (I J G E A C B F D H I) 2150

0 (I J G B A C E F D H I) 1875 *

0 (I J G B A C E F D H I) 1875

0 (I J G B C A E F D H I) 2150 *

0 (I J G A C B E F D H I) 2400 *

Task 8 Code:

```
;-------------------------------------------
; Task 8 Incorporating Mutation

( defmethod mutate ( ( i individual ) &aux mutation )
    ( setf mutation ( mutation ( individual-route i ) ) )
    ( make-instance 'individual
        :number ( individual-number i )
        :route mutation
        :fitness ( fitness mutation )
    )
)

( defconstant *pc-m* 50)

( defmethod maybe-mutate ( ( i individual ) )
    ( if ( <= ( + 1 ( random 100 ) ) *pc-m* )
        ( mutate i )
        i
    )
)

( defmethod mutate-demo ()
    ( setf i ( random-individual ) )
    ( display i )
```

```
    ( dotimes ( x 20 )
        ( setf i ( mutate i ) )
        ( display i )
    )
)

( defmethod maybe-mutate-demo ()
    ( setf i ( random-individual ) )
    ( display i )
    ( dotimes ( x 20 )
        ( setf n ( maybe-mutate i ) )
        ( display-nnl n )
        ( if ( not ( equal n i ) ) ( princ " *" ) )
        ( terpri )
        ( setf i n )
    )
)
```

Task 9: Copy

Task 9 Demo:

Task 9 Code:

```
;-----------------------------------------------------------
; task 9: Copy

( setf *copy-demo* nil )

( defconstant *pc-c* 40 )

( defmethod perform-copies ( ( cp population ) ( np population ) )
    ( dotimes ( i ( nr-copies ) )
        ( perform-one-copy cp np )
    )
)

( defmethod nr-copies ()
    ( * ( / *pc-c* 100 ) *population-size* )
)

( defmethod perform-one-copy ( ( cp population ) ( np population )
    &aux x m mm new-i )
    ( setf m ( select-individual cp ) )
```

```
    ( if *copy-demo* ( format t "Selected individual = ~%" ) )
    ( if *copy-demo* ( display m ) )
    ( setf mm ( maybe-mutate m ) )
    ( if *copy-demo* ( format t "Possibly muted individual = ~&" ) )
    ( if *copy-demo* ( display mm ) )
    ( setf ( individual-number mm ) ( + 1 ( size np ) ) )
    ( if *copy-demo* ( format t "Renumbered individual = ~&" ) )
    ( if *copy-demo* ( display mm ) )
    ( setf new-i ( new-individual ( + 1 ( size np ) ) ( individual-rbg-string mm
) ) )
    ( setf
        ( population-individuals np )
        ( append ( population-individuals np ) ( list new-i ) )
    )
    nil
)

( defmethod empty-population ( ( cp population ) &aux np )
    ( setf np ( make-instance 'population ) )
    ( setf ( population-individuals np ) () )
    ( setf ( population-generation np ) ( + 1 ( population-generation cp ) ) )
    np
)

( defmethod perform-copies-demo ( &aux cp np )
    ( setf cp ( initial-population ) )
    ( setf np ( empty-population cp ) )
    ( format t "-----------------------------------------------------------------
-")
    ( display np )
    ( format t "~%~%-----------------------------------------------------------------
-")
    ( setf *select-demo* t )
    ( setf *copy-demo* t )
    ( dotimes ( i 10 )
        ( perform-one-copy cp np )
        ( format t "-----------------------------------------------------------------
----")
        ( display np )
        ( format t "~%~%-----------------------------------------------------------------
----")
    )
    ( setf *select-demo* nil )
    ( setf *copy-demo* nil )
    nil
```

```
)
```

Task 10: Crossover

Task 10 Demo:

Task 10 Code:

```
;-------------------------------------------------------------
; Task 10: Crossover

( setf *crossover-demo* nil )

( defconstant *pc-x* 60 )

( defmethod perform-crossovers ( ( cp population ) ( np population ) )
    ( dotimes ( i ( nr-crossovers ) )
        ( perform-one-crossover cp np )
    )
)

( defmethod nr-crossovers ()
    ( * ( / *pc-x* 100 ) *population-size* )
)

( defmethod perform-one-crossover ( ( cp population ) ( np population ) )
    ( let ( x m mm mother father new-i )
        ( setf mother ( select-individual cp ) )
        ( setf father ( select-individual cp ) )
        ( if *crossover-demo* ( format t "Selected mother = ~%" ) )
        ( if *crossover-demo* ( display mother ) )
        ( if *crossover-demo* ( format t "Selected father = ~&" ) )
        ( if *crossover-demo* ( display father ) )
        ( setf m ( crossover mother father ) )
        ( if *crossover-demo* ( format t "the crossover = ~&" ) )
        ( if *crossover-demo* ( display m ) )
        ( setf mm ( maybe-mutate m ) )
        ( if *crossover-demo* ( format t "the possibly mutated individual = ~&" )
)
        ( if *crossover-demo* ( display mm ) )
        ( setf ( individual-number mm ) ( + 1 ( size np ) ) )
        ( if *crossover-demo* ( format t "the renumbered individual = ~&" ) )
        ( if *crossover-demo* ( display mm ) )
```

```
        ( setf new-i ( new-individual ( + 1 ( size np ) ) ( individual-rbg-string
mm ) ) )
        ( setf
            ( population-individuals np )
            ( append ( population-individuals np ) ( list new-i ) )
        )
    )
    nil
)

( defmethod crossover ( ( mother individual ) ( father individual )
    &aux mi fi x i )
    ( setf mi ( individual-rbg-string mother ) )
    ( setf fi ( individual-rbg-string father ) )
    ( setf x ( crossover mi fi ) )
    ( setf i ( new-individual 0 x ) )
    i
)

( defmethod perform-crossovers-demo ( &aux cp np )
    ( setf cp ( initial-population ) )
    ( setf np ( empty-population cp ) )
    ( format t "----------------------------------------------------------------------
-")
    ( display np )( format t "~%~%----------------------------------------------------
---------------")
    ( setf *select-demo* t )
    ( setf *crossover-demo* t )
    ( dotimes ( i 10 )
        ( perform-one-crossover cp np )
        ( format t "----------------------------------------------------------------------
----")
        ( display np )
        ( format t "~%~%------------------------------------------------------------
----")
    )
    ( setf *select-demo* nil )
    ( setf *crossover-demo* nil )
    nil
)
```

Task 11: The GA

Task 11 Demo:

Generation 0 population ...

1 (E A C F B H G D J I E) 1800

2 (E F I H A J C G B D E) 2300

3 (G F B A C I E D J H G) 1700

4 (F A E H D J G C I B F) 2100

5 (D A G J F I B H C E D) 2200

6 (A G D J F B H E C I A) 2100

7 (J F I B G C A H D E J) 2400

8 (B A E I D G H J C F B) 2050

9 (B C E J A H I F D G B) 2625

10 (G A D E B I H J F C G) 2700

11 (D G B A F H C J I E D) 2600

12 (F H G A D J B C I E F) 1925

13 (G H C B E A I J D F G) 2675

14 (F A J H I D E B C G F) 3350

15 (H E D G J I C A B F H) 2600

16 (A D E G J F H B C I A) 3350

17 (B D F H C A J G I E B) 2300

18 (E F I A D B H C G J E) 2000

19 (F E I J G B C D H A F) 1900

20 (D H C B A E F J I G D) 2150

21 (B I F J H C D A G E B) 2700

22 (E C I G F J D B A H E) 2800

23 (D F B A E C I H G J D) 2050

24 (C B G A H D F J I E C) 2450

25 (H E F G D A B I J C H) 2050

26 (B F A E H I C J G D B) 2250

27 (J B D F G I H A C E J) 3300

28 (G A H C D J B I E F G) 2825

29 (F A D C H I J E B G F) 2250

30 (G C B F I A J H D E G) 2450

31 (G A B J F C I H D E G) 1850

32 (D H G J I A F E C B D) 2250

33 (H C F J I G E D A B H) 2950

34 (J A C H B F G I E D J) 2500

35 (A D F G C H B E I J A) 2550

36 (D E G H F B I J C A D) 3000

37 (G F D A B E H C I J G) 2100

38 (E F I G B A D J H C E) 2350

39 (C H B J E F I D A G C) 2150

40 (C G A H B E D J I F C) 2900

41 (B H F I D A C J G E B) 2750

42 (G E A J F H C I D B G) 2050

43 (E F B C D I G H A J E) 2900

44 (A D B G E C J F H I A) 2750

45 (D J A G F E H B I C D) 3150

46 (E C I D H B J F G A E) 2225

47 (E D F A G I J C H B E) 2500

48 (A J E H D B I F C G A) 1900

49 (H D F A I B J G C E H) 2350

50 (D F I J E G B A H C D) 2500

51 (B G I F A J E C H D B) 2300

52 (E I A G J F D C B H E) 2175

53 (D F A J C B E H G I D) 2600

54 (E I A B H F G C D J E) 2825

55 (H F A D B C I J E G H) 2250

56 (D H J I E A C B F G D) 1750

57 (D H E J C I G A B F D) 2750

58 (J E D B F H C A G I J) 2750

59 (A D B C G H I J F E A) 3300

60 (A B I C H J E G D F A) 2000

61 (G D H C A I B J F E G) 1800

62 (J F G A D C E B H I J) 2125

63 (A F I J B D E G C H A) 2925

64 (B F G C H I D E A J B) 2850

65 (J F D B C A E G H I J) 1900

66 (B A I E G C H J F D B) 2575

67 (E A B H F C I D G J E) 2550

68 (G D A I F J E C B H G) 1800

69 (J F G B C E H A I D J) 2850

70 (E B H G C A I D F J E) 2825

71 (H D G C B A E I J F H) 2000

72 (A B D J I G E C H F A) 3000

73 (B I C J H A D G E F B) 1900

74 (B A D H G C E F I J B) 1425

75 (D B H J I C F A G E D) 2750

76 (E I H J A F C G D B E) 3050

77 (D J A F G H C I E B D) 2950

78 (J A B F D H I C E G J) 2300

79 (A J H E D G I F B C A) 3050

80 (F J I B A E H G D C F) 2600

81 (D G A J C F E H I B D) 2500

82 (C D B I A H G J E F C) 2275

83 (G B A J I F D E H C G) 2200

84 (B A E C I G J F D H B) 2050

85 (E A J G D H F C I B E) 3000

86 (F H G A B E D J I C F) 2300

87 (F G B E J C D H I A F) 2850

88 (G I F A H B J E C D G) 2475

89 (H I G C F E A B J D H) 2550

90 (D C B A F E H I G J D) 2350

91 (E C B J A H D I G F E) 2350

92 (A C E G F D B J I H A) 2625

93 (B A E C H G F J D I B) 2600

94 (B H G C D A J F E I B) 2600

95 (A H I J C G F D E B A) 3150

96 (H A G D I E F J B C H) 2600

97 (A J F B C H E I G D A) 2300

98 (A J I E B D F H C G A) 2700

99 (B I A H J G F C D E B) 2875

100 (G E I D C F A B H J G) 2200

average fitness of population 0 = 2461.0

*** - >=:

(#<INDIVIDUAL #x1AA779F5> #<INDIVIDUAL #x1AA77ED5> #<INDIVIDUAL #x1AA77B2D> #<INDIVIDUAL #x1AA77F05>

#<INDIVIDUAL #x1AA77E75> #<INDIVIDUAL #x1AA781BD> #<INDIVIDUAL #x1AA77F4D> #<INDIVIDUAL #x1AA7806D>)

is not a real number

Task 11 Code:

```
;------------------------------------------------------------
; Task 11 The GA

;; THE NEXT GENERATION METHOD FOR THE GA

( defmethod next-generation ( ( cp population ) &aux np )
    ( setf np ( empty-population cp ) )
    ( perform-copies cp np )
    ( perform-crossovers cp np )
    np
)

;; THE GA!
( defconstant *nr-generations* 25 )

( defmethod ga ( &aux p )
    ( setf p ( initial-population ) )
    ( terpri )
    ( summarize p )
    ( dotimes ( i *nr-generations* )
        ( setf p ( next-generation p ) )
        ( check-average p )
    )
    ( terpri )
    ( display ( most-fit-individual ( population-individuals p ) ) )
)

;; METHODS TO PROVIDE INFORMATION ON "PROGRESS"
( defmethod summarize ( ( p population ) )
    ( display p )
    ( check-average p )
    ( terpri )
)

( defmethod check-average ( ( p population ) )
    ( format t "average fitness of populatioon ~A = ~A~%"
        ( population-generation p )
        ( average p )
    )
)
```